SpecVer & LEM'12 projects XX (2013) 1–10 1
Dombrova Research

Brulion – this version for your eyes only
Dowód poprawnośći algorytmu Winograda

Andrzej Salwicki
salwicki@mimuw.edu.pl
Dąbrowa Leśna

Streszczenie. Przedstawiamy algorytm Winograda i dowód jego poprawności.

1. Wprowadzenie

Algorytm Winograda może być przydatny w obliczaniu iloczynu macierzy kwadratowych, szczególnie gdy elementami macierzy są obiekty reprezentujące jakieś ciało inne od ciała liczb rzeczywistych. Tzn. w przypadku gdy dane ciało $\mathcal{C}$ jest zaimplementowane w programie przez klasę C.

Drugim i ważniejszym powodem do skreślenia tej notatki jest potrzeba zwrócenia uwagi na znikomą przydatność asercji i tzw. programowania przez kontrakt (*ang.* design by contract). Asercje są przydatne jako notatki, ale nie stanowią rozwiązania problemu zapewnienia poprawności algorytmu.

Zapraszam do czytania, zwłaszcza rozdziału Dowód poprawności.

2. Algorytm

W tym rozdziale pojawiają się dwa warunki:

- warunek wstepny – Precondition,
- warunek końcowy – Postcondition

oraz algorytm – algorytm Winograda.

Jak się upewnić, że algorytm jest poprawny ze względu na warunek poczatkowy i warunek końcowy? W literaturze proponowane są dwa podejścia:

- udowodnij częściową porawność sprawdzając pewne niezmienniki – jest to tzw. metoda Floyda-Hoare'a,

- wykonaj pewną liczbę obliczeń testowych i zaufaj, że w trakcie eksploatacji algorytmu nie okaże się, że jest on niepoprawny.

W obliczeniach testowych można w trakcie obliczeń sprawdzać wcześniej wstawione warunki – asercje. Czy rzeczywiście są one pomocne?

W następnym rozdziale pokażemy inną drogę - drogę dowodzenia lematów i w końcu twierdzenia o poprawności (całkowitej) algorytmu Winograda względem warunku początkowego **Precondition** i warunku końcowego **Postcondition**.

```
signal Niezgoda;
unit Winograd : procedure(A, B : array_of array_of real; output C : array_of array_of real);
Precondition: wymagaj by A i B były macierzami kwadratowymi rozmiaru n x n
Postcondition: zapewnij, że obliczona macierz C jest produktem macierzy A i B, C = A * B
var i, j, k, n, m : integer, W, V : array_of real, p : boolean, s : real;
begin
{ ustalić czy macierze mogą być mnożone tzn. czy ilość wierszy w A = ilosc kolumn w B? }
{ ustalić czy n jest parzyste? }
{ obliczyc preprocessing }
--------------------------------------------------------------------------------
{ dynamiczne sprawdzanie precondition }
if lower(A) ≠ lower(B) or lower(A)≠ 1 or upper(A)≠ upper(B) then
   raise Niezgoda
fi;
i := upper(A);
j := lower(A);
n := i-j+1;
for l := j to i do
   if lower(A(l)) ≠ lower(B(l)) or lower(A(l))≠ 1 or upper(A(l))≠ upper(B(l) or upper(A(l)≠upper(A)
   then
      raise Niezgoda
   fi;
od;
Assertion sprawdzono: macierze są kwadratowe, rozmiaru n x n
--------------------------------------------------------------------------------
{ można mnożyć }
p := (n mod 2) = 0;
m := n div 2;
array W dim(1 : n);
array V dim(1 : n);
array C dim(1 : n);
for i := 1 to n do
   array C(i) dim(1 : n)
od;
```

```
{ obliczanie "preprocessingu"}
for j:= 1 to n do
   s:=0;
   for i := 1 to m do
      s := A[j, 2 * i - 1] * A[j, 2 * i] + s;
   od;
   W[j] := s;
od;
```

Assertion 1: Dla każdego $j, 1 \leq j \leq n$, $\quad W_j = \sum_{i=1}^{n \div 2} A_{j,2i-1} * A_{j,2i}$

```
for j:= 1 to n do
   s:=0;
   for i := 1 to m do
      s := B[2*i-1,j] * B[2*i,j] +s;
   od;
   V[j] := s;
od;
```

Assertion 2: Dla każdego $j, 1 \leq j \leq n$, $\quad V_j = \sum_{i=1}^{n \div 2} B_{2i-1,j} * B_{2*i,j}$

```
{obliczanie iloczynu macierzy }
for i := 1 to n do
   for j := 1 to n do
      s:= 0;
      for k := 1 to m do
         s:= (A[i,2*k-1]+B[2*k,j]) * (B[2*k-1,j]+A[i,2*k]) +s;
      od;
```

Assertion 3: $\forall_{1 \leq i \leq n}, \forall_{1 \leq j \leq n}$, $\quad s = \sum_{i=1}^{n \div 2} (A_{i,2k-1} + B_{2k,j}) * (B_{2*k-1,j} + A_{i,2k})$

```
      C[i, j] := s - W[i] - V[j];
```

Assertion 4: $\forall_{1 \leq i \leq n}, \forall_{1 \leq j \leq n}$ $\quad C_{i,j} = \sum_{i=1}^{2(n \div 2)} (A_{i,k} * B_{k,j})$

```
      if not p then
         C[i, j] := C[i, j] + A[i, n] * B[n, j]; { poprawiamy - gdy n jest nieparzyste }
      fi;
   od; { j }
od; { i }
```

Assertion 5: Dla każdych wartości $i, j, 1 \leq i \leq n, 1 \leq j \leq n$, $\quad C_{i,j} = \sum_{i=1}^{n} (A_{i,k} * B_{k,j})$

```
end Winograd;
```

3. Dowód poprawności

Naszym zadaniem jest wykazać nastepującą implikację

$$\text{Precondition} \Rightarrow [\text{Algorytm Winograda}]\text{Postcondition}$$

co się czyta tak: *jeśli dane spełniają warunek wstępny to algorytm Winograda kończy obliczenia nie sygnalizując błędu i wyniki spełniają warunek końcowy*.

Sprawdzenie czy warunek wstępny jest spełniony przez dane może być w części wykonane przez kompilator. Kompilator może sprawdzić czy parametry aktualne sa tablicami dwuwymiarowymi. Druga część warunku, że rozmiary tablic są równe $n \times n$ nie może być sprawdzona przed wywołaniem procedury Winograd, nie może też być udowodniona. Wobec tego wykonywanie procedury rozpoczynamy od (dynamicznego) sprawdzania kształtu i rozmiaru tablic[1]. Można rozważać czy nie dałoby się udowodnić, o programie stosującym procedurę Winograd, że warunek wstępny jest spełniony za każdym razem gdy procedura Winograd jest wywoływana w naszym programie. Ale czy można zagwarantować, że każdy program stosujący algorytm Winograda będzie sprawdzał warunek wstępny? lub go dowodził? Lepiej więc zostawić sprawdzanie warunku wstępnego procedurze Winograd.

Do algorytmu Winograd wstawiliśmy asercje. Mają one za zadanie:

1. umożliwić sygnalizację naruszenia warunku asercji w trakcie wykonywania programu,
2. ułatwić argumentację na rzecz tezy o poprawności algorytmu.

W tym przypadku trudno mówić o dynamiczej weryfikacji: ażeby sprawdzić warunek wyliczony w asercji trzeba powtórzyć obliczenia – to niewiele nam daje. Ponadto, tu uwaga natury ogólnej, zamiana asercji na instrukcję warunkową

$$\textbf{if } \mathit{warunek_asercji} \textbf{ then } \text{wrzuć_ wyjątek } \textbf{fi}$$

zapewnia tylko tyle, że podczas wykonywania algorytmu zostanie zasygnalizowany błąd. Nie mamy nawet gwarancji, że zdarzy się to zawsze gdy program zawiera błędy.

Natomiast asercje możemy zastąpić lematami i udowodnić je

Lemat 1. Dla każdego $j, 1 \leq j \leq n$, i $m = n \div 2$ zachodzi

$$K : \left\{ \begin{array}{l} s := 0; \\ \textbf{for } i := 1 \textbf{ to } m \\ \textbf{do} \\ \quad s := A[j, 2*i-1] * A[j, 2*i] + s; \\ \textbf{od}; \\ W[j] := s; \end{array} \right\} \left(W_j = \sum_{i=1}^{n \div 2} A_{j,2i-1} * A_{j,2i} \right)$$

[1] W loglanie'82 dwuwymiarowej tablicy możemy nadać kształt trójkatny, wstęgowy i oczywiście kształt kwadratowy

Lemat 2. Dla każdego $j, 1 \leq j \leq n$, i $m = n \div 2$ zachodzi

$$\left\{\begin{array}{l} s := 0; \\ \textbf{for } i := 1 \textbf{ to } m \\ \textbf{do} \\ \quad s := B[2*i-1,j]*B[2*i,j]+s; \\ \textbf{od}; \\ V[j] := s; \end{array}\right\} \left(V_j = \sum_{i=1}^{n\div 2} B_{2i-1,j} * B_{2i,j} \right)$$

Kolejny lemat

Lemat 3. $\forall_{1 \leq i \leq n}, \forall_{1 \leq j \leq n}$

$$\left\{\begin{array}{l} s := 0; \\ \textbf{for } \text{k:= 1 } \textbf{to} \text{ m} \\ \textbf{do} \\ \quad s := (A[i,2*k-1]+B[2*k,j]) \\ \qquad *(B[2*k-1,j]+A[i,2*k])+s; \\ \textbf{od}; \end{array}\right\} \left(s = \sum_{i=1}^{n\div 2} (A_{i,2k-1} + B_{2k,j}) * (B_{2k-1,j} + A_{i,2k}) \right)$$

Lemat 4.
Przy założeniu, że tablice A i B są macierzami kwadratowymi rozmiaru n x n i że zachodzą lematy 1 oraz 2, prawdziwa jest nastepująca formuła algorytmiczna

$$\forall_{1 \leq i \leq n}, \forall_{1 \leq j \leq n} \left\{\begin{array}{l} s := 0; \\ \textbf{for } k := 1 \textbf{ to } m \\ \textbf{do} \\ \quad s := (A[i,2*k-1]+B[2*k,j]) \\ \qquad *(B[2*k-1,j]+A[i,2*k])+s; \\ \textbf{od}; \\ C[i,j] := s - W[i] - V[j]; \end{array}\right\} \left(s = \sum_{i=1}^{2*(n\div 2)} A_{i,k} * B_{k,j} \right)$$

Lemat 5.
Z prawdziwości lematów 1 i 2 wynika, że następująca formuła jest prawdziwa

$$\left\{\begin{array}{l}
\textbf{for } i := 1 \textit{ to } n\\
\textbf{do}\\
\quad \textbf{for } j := 1 \textit{ to } n\\
\quad \textbf{do}\\
\qquad s := 0;\\
\qquad \textbf{for } \text{k:= 1 } \textbf{to} \text{ m}\\
\qquad \textbf{do}\\
\qquad\quad s := (A[i, 2*k-1] + B[2*k, j])\\
\qquad\qquad *(B[2*k-1, j] + A[i, 2*k]) + s;\\
\qquad \textbf{od};\\
\qquad C[i,j] := s - W[i] - V[j];\\
\qquad \textbf{if not } p\\
\qquad \textbf{then}\\
\qquad\qquad C[i,j] := C[i,j] + A[i,n]*B[n,j]\\
\qquad \textbf{fi};\\
\quad \textbf{od}\\
\textbf{od}
\end{array}\right\}\left(\forall_{1\le i\le n}, \forall_{1\le j\le n}\, C_{i,j} = \sum_{i=1}^{n} A_{i,k} * B_{k,j}\right)$$

4. Dowody lematów

4.1. Dowód lematu 1

W dowodzie wykorzystujemy następującą własność programów **for**:
niech napis $\omega(i)$ oznacza wyrażenie arytmetyczne (zmienna i może, ale nie musi, w nim wystepować)

$$\left\{\begin{array}{l}
s := 0\\
\textbf{for } i := 1 \textbf{ to } n\\
\textbf{do}\\
\quad s := \omega(i) + s\\
\textbf{od}
\end{array}\right\}\left(s = \sum_{i=0}^{n} \omega(i)\right)$$

Pozostaje skorzystać z aksjomatu instrukcji przypisania

$$\left(s = \sum_{i=0}^{n} \omega(i)\right) \Rightarrow \{W[j] := s\} \left(W(j) = \sum_{i=0}^{n} \omega(i)\right)$$

□

Dowody lematów 2 i 3 przebiegają podobnie.

4.2. Dowód lematu 4

Należy udowodnić

$$(s = \sum_{k=1}^{n\div 2}(A_{i,2k-1} + B_{2k,j}) * (B_{2*k-1,j} + A_{i,2k}) \Rightarrow (s - W[i] - V[j] = \sum_{k=1}^{2(n\div 2)} A_{i,k} * B_{k,j})$$

Rozwińmy mnożenie i zastosujmy rozdzielność mnożenia wzgledem dodawania

$$\begin{aligned}
&\sum_{k=1}^{n\div 2}(A_{i,2k-1} + B_{2k,j}) * (B_{2*k-1,j} + A_{i,2k}) \\
&\quad = \sum_{k=1}^{n\div 2}[A_{i,2k-1} * B_{2*k-1,j} + A_{i,2k-1} * A_{i,2k} + B_{2k,j} * B_{2k-1,j} + B_{2k,j} * A_{i,2k}] \\
&\quad = \sum_{k=1}^{n\div 2} A_{i,2k-1} * B_{2*k-1,j} + \sum_{k=1}^{n\div 2} A_{i,2k-1} * A_{i,2k} + \sum_{k=1}^{n\div 2} B_{2k,j} * B_{2k-1,j} + \sum_{=1}^{n\div 2} B_{2k,j} * A_{i,2k}
\end{aligned}$$

Skorzystamy z lematów 1 oraz 2

$$\sum_{k=1}^{n\div 2}(A_{i,2k-1} + B_{2k,j}) * (B_{2*k-1,j} + A_{i,2k}) - W[i] - V[j] = \sum_{k=1}^{n\div 2} A_{i,2k-1} * B_{2*k-1,j} + \sum_{k=1}^{n\div 2} B_{2k,j} * A_{i,2k}$$

Wykorzystujemy przemienność mnożenia i łączność dodawania

$$\sum_{k=1}^{n\div 2} A_{i,2k-1} * B_{2*k-1,j} + \sum_{k=1}^{n\div 2} B_{2k,j} * A_{i,2k} = \sum_{k=1}^{2(n\div 2)} A_{i,k} * B_{k,j}$$

□

4.3. Dowód lematu 5

Trzeba wykazać, że

$$(s = \sum_{k=1}^{2(n\div 2)} A_{i,k} * B_{k,j}) \Rightarrow \left\{ \begin{array}{l} \textbf{if not } p \\ \textbf{then} \\ \qquad C[i,j] := C[i,j] + A[i,n] * B[n,j] \\ \textbf{fi}; \end{array} \right\} (s = \sum_{k=1}^{n} A_{i,k} * B_{k,j})$$

Pamiętamy, że p= (n mod 2 =0). Jeżeli n jest liczbą parzystą to $2(n \div 2) = n$.

$$C_{i,j} = \sum_{i=1}^{n} A_{i,k} * B_{k,j}$$

W przeciwnym przypadku (tzn. gdy **not** p) sumowanie zakonczyło się dla $k = n - 1$. Trzeba więc dodać wartość $A[i,n] * B[n,j]$

□

5. Uwagi końcowe

1. Zauważ, że algorytm ten pozwala mnożyć macierze liczb zespolonych, ... Zmieńmy nagłówek procedury

unit Winograd_Generic: **procedure**(**type** T, **function** add(x,y:T):T, **function** mult(x,y:T):T,
function subt(x,y:T):T, A, B: **array_of array_of** T, output C: **array_of array_of** T);

i odpowiednio w treści procedury zapiszmy add(u,v) zamiast u+v, mult(u,v) zmiast u*v, subt(u,v) zamiast u-v.,

Nasz warunek wstępny przybiera teraz inną postać

Precondition: Argumenty procedury Winograd_Generic spełniaja następujące warunki

1. pierwszy argument jest nazwą klasy,
2. kolejne trzy argumenty są nazwami funkcji, dwuargumentowych ze zbioru |T| obiektów typu T w zbiór |T|,
3. następne dwa argumenty to nazwy tablic kwadratowych jednakowego rozmiaru n x n,
4. ostatni argument także jest nazwą tablicy, tablica ta zostanie przekazana wywołującemu procedurę Winograd_Generic,
5. zbiór |T| obiektów klasy T wraz z operacjami add, mult i subt jest *pierścieniem*.

Postcondition: Algorytm zwraca tablicę C obiektów typu T, taką, że dla każdych i, j, $1 \leq i \leq n, 1 \leq j \leq n$

$$C_{i,j} = \lceil add \rceil_{k=0}^{n} \, mult(A_{i,k}, B_{k,j})$$

Poprzednia wersja dowodu lematu 1
Niech j będzie liczbą naturalną taką, że $1 \le j \le n$, niech $m = n \div 2$. Wykażemy, po wykonaniu instrukcji ujętych w nawiasy zachodzi warunek $W_j = \sum_{i=1}^{n\div 2} A_{j,2i-1} * A_{j,2i}$.
Przypadek $m = 0$. Jesli m jest równe zero to na mocy aksjomatu instrukcji for

$$\boxed{\textbf{for } i := 1 \textbf{ to } 0 \; do \; \mathbf{K} \; od \, \alpha \Leftrightarrow \alpha}$$

mamy równoważną jej formułę,

$$\left\{ \begin{array}{l} s := 0; \\ W[j] := s; \end{array} \right\} \left(W_j = \sum_{i=1}^{n\div 2} A_{j,2i-1} * A_{j,2i} \right)$$

czyli

$$\{s := 0; W[j] := s; \} \left(W_j = \sum_{i=1}^{n\div 2} A_{j,2i-1} * A_{j,2i} \right)$$

Jeśli dwukrotnie zastosujemy aksjomat instrukcji przypisania to otrzymamy najpierw

$$\{s := 0; \} \left(s = \sum_{i=1}^{n\div 2} A_{j,2i-1} * A_{j,2i} \right)$$

a potem

$$\left(0 = \sum_{i=1}^{n\div 2} A_{j,2i-1} * A_{j,2i} \right)$$

Równocześnie z własności sumy Σ wyrażenie po prawej znaku = ma wartość zero, co kończy analizę przypadku $m = 0$ ponieważ wszystkie te formuły sa sobie równoważne.

Krok indukcyjny Załóżmy, że nasz lemat zachodzi dla wartości m równej k. Wykażemy, że zachodzi on też dla $m = k + 1$. Rozważmy formułę

$$\left\{ \begin{array}{l} s := 0; \\ \textbf{for } i := 1 \textbf{ to } k+1 \\ \textbf{do} \\ \quad s := A[j, 2*i-1] * A[j, 2*i] + s; \\ \textbf{od}; \\ W[j] := s; \end{array} \right\} \left(W_j = \sum_{i=1}^{k+1} A_{j,2i-1} * A_{j,2i} \right)$$

Wykorzystamy inny aksjomat instrukcji for.

$$\{\textbf{for}\ i := 1\ \textbf{to}\ n+1\ do\ \mathbf{K}\ \textbf{od}\,\}\,\alpha \Leftrightarrow \{\textbf{for}\ i := 1\ \textbf{to}\ n\ do\ \mathbf{K}\ \textbf{od}; i := n+1; \mathbf{K}\}\ \alpha$$

Z założenia indukcyjnego wiemy, że

$$\left\{\begin{array}{l} s := 0; \\ \textbf{for}\ i := 1\ \textbf{to}\ k \\ \textbf{do} \\ \quad s := A[j, 2*i-1] * A[j, 2*i] + s; \\ \textbf{od}; \end{array}\right\} \left(s = \sum_{i=1}^{k} A_{j,2i-1} * A_{j,2i}\right)$$

Pozostaje do wykazania, że

$$\left(s = \sum_{i=1}^{k} A_{j,2i-1} * A_{j,2i}\right) \Rightarrow \{i := k+1; s := A[j, 2*i-1] * A[j, 2*i] + s\}(s = \sum_{i=1}^{k+1} A_{j,2i-1} * A_{j,2i})$$

... UZUPEŁNIJ!

□